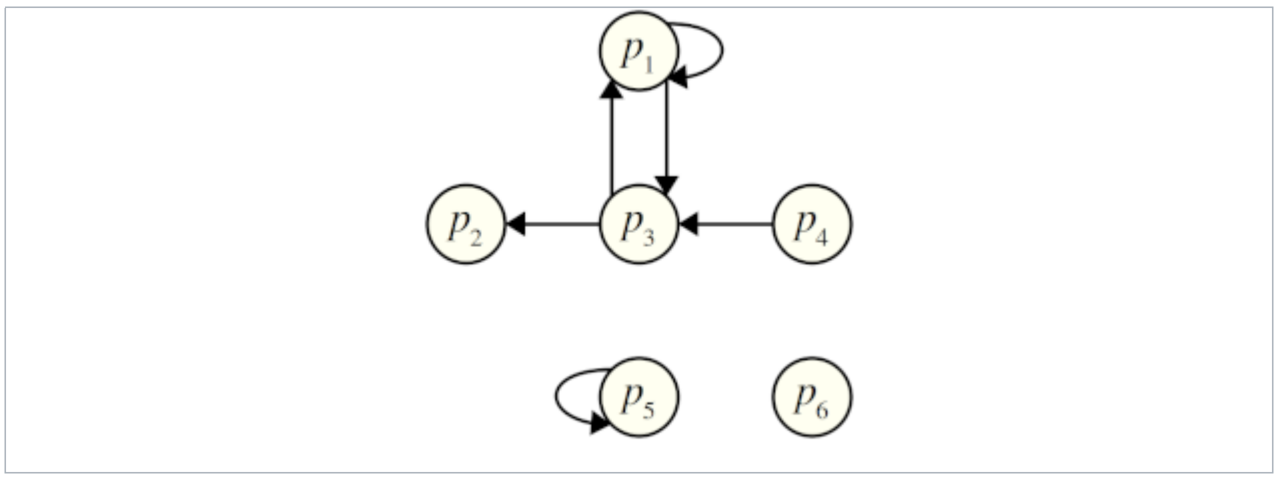
PSET2, Part 1: Logical Expressions in English and in Code

👈 *Back to problem set index.*

Problem One: Interpersonal Dynamics

Consider the following diagram:



If there's an arrow from a person x to a person y, then person x loves person y. We'll denote this by writing $Loves(x, y)$. For example, in this picture, we have $Loves(p_4, p_3)$ and $Loves(p_5, p_5)$, but not $Loves(p_4, p_1)$.

There are no "implied" arrows in this diagram. For example, although p_1 loves p_3 and p_3 loves p_2, the statement $Loves(p_1, p_2)$ is false because there's no arrow from p_1 to p_2. Similarly, even though p_4 loves p_3, the statement $Loves(p_3, p_4)$ is false because there's no arrow from p_3 to p_4.

Below is a series of first-order logic statements. Some are true, and some are false. Your task is, for each false statement, to tell us the smallest collection of arrows that need to be added to the diagram in order to make the statement true.

This problem is autograded. Download the starter files for Problem Set Two and extract them somewhere convenient. You'll enter your answers into the file `res/Interpersonal.dynamics`. Specifically, do the following:

- For each true statement, answer `true`.
- For each false statement, tell us who needs to love whom to make the formula true. Your answer needs to use the smallest number of additional loves relations to receive full credit.

You can use the local "Run Tests" button to check your work locally, or submit to Gradescope. As with Problem Set One, you can submit as many times as you'd like; we'll only grade your last submission.

We've included answers to the first three of these questions as a reference; you need to fill in the rest.

i. $Loves(p_1, p_3)$

ii. $Loves(p_3, p_4)$

iii. $Loves(p_1, p_2) \land Loves(p_2, p_1)$

iv. $Loves(p_1, p_2) \lor Loves(p_2, p_1)$

v. $Loves(p_1, p_1) \to Loves(p_5, p_5)$

vi. $Loves(p_1, p_2) \to Loves(p_4, p_3)$

vii. $Loves(p_1, p_3) \to Loves(p_3, p_6)$

viii. $Loves(p_1, p_4) \to Loves(p_4, p_5)$

ix. $Loves(p_1, p_4) \leftrightarrow Loves(p_2, p_3)$

x. $Loves(p_1, p_3) \leftrightarrow Loves(p_5, p_5)$

xi. $\forall x.\ \exists y.\ Loves(x, y)$

xii. $\forall x.\ \exists y.\ Loves(y, x)$

xiii. $\forall x.\ \exists y.\ (x \neq y \land Loves(x, y))$

xiv. $\forall x.\ \exists y.\ (x \neq y \land Loves(y, x))$

xv. $\exists x.\ \forall y.\ Loves(x, y)$

xvi. $\exists x.\ \forall y.\ (x \neq y \to Loves(x, y))$

Problem Two: Propositional Completeness

In this problem, you'll explore some redundancies within the language of propositional logic.

This problem is autograded. Edit `res/PropositionalCompleteness.proplogic` with your answers. There's information inside the file with information about how to structure your answer. Briefly, if the online Truth Table Tool can understand your answer, so can our autograder. As usual, feel free to submit as many times as you'd like; we'll only grade your last submission.

In lecture, we covered the seven propositional connectives, which for convenience we've listed below:

$$\land \quad \lor \quad \neg \quad \to \quad \leftrightarrow \quad \top \quad \bot$$

We settled on this set of connectives because they're convenient and expressive. However, it turns out that we could get away with fewer connectives than this.

i. Write expression equivalent to $\bot$ that does not use any connectives besides $\land$, $\lor$, $\neg$, and $\top$. (You're welcome to use parentheses, but do not use any variables.)

ii. Write an expression equivalent to $p \to q$ that does not use any connectives besides $\land$, $\lor$, $\neg$, and $\top$. (You're welcome to use the variables p and q, along with parentheses.)

iii. Write an expression equivalent to $p \leftrightarrow q$ that does not use any connectives besides $\land$, $\lor$, $\neg$, and $\top$. (You're welcome to use the variables p and q, along with parentheses.)

Your answers to parts (i), (ii), and (iii) of this problem show that the the four propositional connectives $\land$, $\lor$, $\neg$, and $\top$ collectively are ***sufficient*** – the other three connectives can be rewritten purely in terms of them. However, there's some redundancy within those four connectives, and we can express all propositional formulas just using three of them.

iv. Write an expression equivalent to $p \lor q$ that does not use any connectives besides $\land$, $\neg$, and $\top$. (You're welcome to use the variables p and q, along with parentheses.)

We can push this further. You can rewrite any propositional formula using just the $\to$ and $\bot$ connectives!

v. Write an expression equivalent to $\top$ that does not use any connectives besides $\to$ and $\bot$. (You're welcome to use parentheses, but do not use any variables.)

vi. Write an expression equivalent to $\neg p$ that does not use any connectives besides $\to$ and $\bot$. (You're welcome to use the variable p, along with parentheses.)

vii. Write an expression equivalent to $p \land q$ that does not use any connectives besides

$\to$ and $\bot$. (You're welcome to use the variables p and q, along with parentheses.)

As a hint, what happens if you negate an implication?

To recap: given the $\to$ and $\bot$ connectives, you can express $\land$, $\neg$, and $\top$. From $\land$, $\neg$, and $\top$ you can get $\land$, $\lor$, $\neg$, and $\top$. And from those four connectives, you can get back the original seven. Overall, any propositional formula can be expressed purely in terms of $\to$ and $\bot$. Nifty!

Problem Three: Executable Logic

This problem, and the ones after it, concern worlds populated by cats, robots, and humans. Love is in the air, and it seems appropriate to pin down the group dynamics using first-order logic. Let's assume we have the predicates

- $Person(p)$, which states that p is a person;
- $Cat(c)$, which states that c is a cat;
- $Robot(r)$, which states that r is a robot; and
- $Loves(x, y)$, which states that x loves y.

As a note, the $Person$, Cat, and $Robot$ predicates are mutually exclusive. For example, you can't have a robot cat or a cat person. (I mean, you can have a "cat person" in the sense that you can have a person who likes cats, but not someone who is literally both a cat and a person. 😃) Additionally, you can assume that each entity in the world is either a person, a cat, or a robot. Finally, note that love is not necessarily symmetric. It's possible for A to love B but not the other way around. (Alas!)

Below is a list of six first-order logic statements. Your task is to implement the six C++ functions defined in the file `src/ExecutableLogic.cpp`, each of which determines whether one of the first-order logic formulas is true about a set of robots, cats, and people. Each robot, cat, or person is represented using a variable of type `Entity`, and we've provided the following C++ functions to you, which mirror the four predicates given above:

```
bool Person(Entity p);
bool Cat   (Entity c);
bool Robot (Entity r);
bool Loves (Entity x, Entity y);
```

Our provided starter files will run the six functions you'll implement on some sample worlds, which you can use to test out your implementations.

i. Consider the following first-order logic formula:

$$\exists x.\ Cat(x)$$

Write C++ code for a function

```
bool isFormulaTrueFor_partI(std::set<Entity> S)
```

that takes in a set S and returns whether the above formula is true for the entities in that set.

ii. Repeat the above exercise with this first-order logic formula:

$$\forall x.\ Robot(x)$$

iii. Repeat the above exercise with this first-order logic formula:

$$\exists x.\ (Person(x) \land Loves(x, x))$$

iv. Repeat the above exercise with this first-order logic formula:

$$\forall x.\, (Cat(x) \to Loves(x, x))$$

v. Repeat the above exercise with this first-order logic formula:

$$\begin{array}{l} \forall x.\, (Cat(x) \to \\ \quad \exists y.\, (Person(y) \land \neg Loves(x, y)) \\ ) \end{array}$$

It's a lot easier to write code for this one if you use a helper function.

vi. Repeat the above exercise with this first-order logic formula:

$$\begin{array}{l} \exists x.\, (Robot(x) \leftrightarrow \\ \quad \forall y.\, Loves(x, y) \\ ) \end{array}$$

Problem Four: First-Order Negations

For each of the first-order logic formulas below, find a first-order logic formula that is the negation of the original statement. Your final formula must not have any negations in it except for direct negations of predicates. For example, given the formula

$$\begin{array}{l} \forall c.\, (Cat(c) \to \\ \quad \exists p.\, (Person(p) \land Loves(p, c)) \\ ), \end{array}$$

you could give the formula

$$\begin{array}{l} \exists c.\, (Cat(c) \land \\ \quad \forall p.\, (Person(p) \to \neg Loves(p, c)) \\ ). \end{array}$$

However, you couldn't give as an answer the formula

$$\begin{array}{l} \exists c.\, (Cat(c) \land \\ \quad \neg \exists p.\, (Person(p) \land Loves(p, c)) \\ ), \end{array}$$

since the inner negation could be pushed deeper into the expression.

To submit your answers, edit the file `res/FirstOrderNegations.fol` with your final formulas. That file contains information about how to format your answers.

We ***strongly recommend*** reading over the Guide to Negations before starting this problem.

i. Fully negate this formula:

$$\begin{array}{l} \forall p.\, (Person(p) \to \\ \quad \exists c.\, (Cat(c) \land Loves(p, c) \land \\ \quad\quad \forall r.\, (Robot(r) \to \neg Loves(c, r)) \\ \quad) \\ ) \end{array}$$

ii. Fully negate this formula:

$$\begin{array}{l} (\forall x.\, (Person(x) \leftrightarrow \exists r.\, (Robot(r) \land Loves(x, r)))) \\ \quad \to \\ (\forall r.\, \forall c.\, (Robot(r) \land Cat(c) \to Loves(r, c))) \end{array}$$

Be careful - make sure you understand how that formula is parenthesized before you try negating it.

iii. Fully negate this formula:

$$\forall c.\,(Cat(c) \to \\ \quad \exists r.\,(Robot(r) \land \\ \quad\quad \forall x.\,(Loves(c, x) \leftrightarrow r = x) \\ \quad) \\ )$$

iv. Fully negate this formula:

$$\exists x.\,(Cat(x) \land \\ \quad (\forall r.\,(Loves(r, x) \to Robot(r)) \lor \forall p.\,(Loves(p, x) \to Person(p))) \\ )$$

Problem Five: This, But Not That

Below is a series of pairs of statements about groups of cats, robots, and people. For each pair, find the ***absolute simplest world*** in which the first statement is true and the second statement is false. (By “absolute simplest,” we mean using as few entities as possible, and, of solutions with the fewest entities possible, having as few entities love each other as possible.)

To submit your answers, edit the file **`res/ThisButNotThat.worlds`**. There’s information in that file about how to specify your worlds.

	Make this statement true…	**… and this statement false**
i	$\forall y.\,\exists x.\,Loves(x, y)$	$\exists x.\,\forall y.\,Loves(x, y)$
ii	$\forall x.\,(Person(x) \lor Cat(x))$	$(\forall x.\,Person(x)) \lor (\forall x.\,Cat(x))$
iii	$(\exists x.\,Robot(x)) \land (\exists x.\,Loves(x, x))$	$\exists x.\,(Robot(x) \land Loves(x, x))$
iv	$(\forall x.\,Cat(x)) \to (\forall y.\,Loves(y, y))$	$\forall x.\,\forall y.\,(Cat(x) \to Loves(y, y))$
v	$\exists x.\,(Robot(x) \to \forall y.\,(Robot(y)))$	$(\forall x.\,Robot(x)) \lor (\forall x.\,\neg Robot(x))$

As a hint, if you want to make a statement false, make its negation true.

For part (i), remember that you need to give the simplest possible world where "this" is true and "that" is false. As a note, it is possible to end up with a world where there is no way to simplify that specific world by removing any person or relationship from the world, even though it's not as small as possible. An analogy: suppose you were asked to make a stable chair with as few legs as possible. If you came up with a four-legged chair, there's no way for you to remove any of the legs so that the resulting chair would be stable. However, by fundamentally changing the design and making the chair a tripod, you can indeed create a stable, three-legged chair.

For part (v), does the statement in the left column look fishy to you?

Problem Six: Translating into Logic

In each of the following, write a statement in first-order logic that expresses the indicated sentence. Your statement may use any first-order construct (equality, connectives, quantifiers, etc.), but you ***must*** only use the predicates $Person$, $Robot$, Cat, and $Loves$.

To submit your answers, edit the file **`res/TranslatingIntoLogic.fol`** with your formulas. There’s information in that file about the expected format for your answers.

Please read the [Guide to Logic Translations] before starting this problem.

i. Write a statement in first-order logic that says “robots do not love.” (How sad!)

As a reminder, love is considered directional. Even if robots do not love, it’s possible that people or cats might love robots. For example, I could love my

Randomly even if it feels nothing toward me.

ii. Write a statement in first-order logic that says “each robot loves every cat, but no cat loves any person.”

iii. Write a statement in first-order logic that says “each cat only loves itself.” (Okay, I’m not that cynical about cats. But it’s still a good exercise to translate this statement!)

iv. Write a statement in first-order logic that says “if you pick a person, you’ll find that they love a cat if and only if they also love a robot.”

Watch your operator precedence.

v. Write a statement in first-order logic that says “each person loves exactly two cats and nothing else.” To clarify, each person is allowed to love a different pair of cats.

vi. Write a statement in first-order logic that says “no two robots love exactly the same set of cats.”

As a reminder, you're restricted to just using the predicates we provided you, so you can't use the ∈ predicate or the **Set** *predicate like we did in lecture. You'll need to find another way to express this idea. Check the Guide to Logic Translations' section on set theory.*

Looking for a good read on the theme of people, robots, pets, and love? Check out Ted Chiang’s novella *The Lifecycle of Software Objects*, which explores these ideas in depth. 😃

Problem Seven: All the Everys and Alls

Many programming languages provide a function that takes as input a group of items and some predicate, then returns whether the predicate is true for all items in the group. For example, JavaScript has `Array.prototype.every`, which can be used like this:

```
function isOdd(n) {
    return n % 2 == 1;
}

[1, 3, 5].every(isOdd)  // true, all these numbers are odd
[1, 2, 3].every(isOdd)  // false, 2 is not odd
[1      ].every(isOdd)  // true, this one number is odd
[2      ].every(isOdd)  // false, this one number is not odd
[       ].every(isOdd)  // true, see below.
```

The last line of the above code sample highlights an interesting detail: if you use `every` on an empty list of values, then `every` returns true regardless of what predicate is provided.

You might think this is just a quirk of JavaScript, but this same behavior can be found in dozens of other languages. For example, C++'s `ranges::all_of`, C♯'s `Enumerable.All`, Haskell's `all`, Ruby's `all?`, Python's `all`, Kotlin's `all`, Swift's `allSatisfy(_:)`, and Common Lisp's `every` all have the same behavior.

There is a concept we covered in lecture that explains why this behavior is the correct way for this function to handle the empty list (and why so many different languages define their functions similarly). What concept is this? Briefly explain your answer; we're looking for something one or two sentences long.

Website programming by Julie Zelenski with minor edits by Keith Schwarz and Sean Szumlanski. This page last updated 2025-Jan-15.